A General Portable Performance Metric

- ***Informally,*** **Time to solve a problem of size, n,**

$$T(n) \text{ is } O(\log n)$$

$$\leftarrow T(n) = c \log_2 n$$

- ***Formally:***
  - **$O(g(n))$ is the set of functions, f, such that**

$$f(n) < c\, g(n)$$

for some constant, $c > 0$, and $n > N$

ie for sufficiently large n

  - **Alternatively, we may write**

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} \le c$$

and say

g is an upper bound for f

A General Portable Performance Metric

- $O(g)$
  - the set of functions that grow no faster than g.
- $g(n)$ describes the worst case behaviour of an algorithm that is $O(g)$
- Two additional notations
- $\Omega(g)$
  - the set of functions, f, such that

  $$f(n) > c\,g(n)$$

  for some constant, c, and $n > N$

  g is a lower bound for f

- $\Theta(g) = O(g) \cap \Omega(g)$

  Set of functions growing at the same rate as g

Properties of the O notation

- Constant factors may be ignored
  - $\forall\ k > 0,\ kf$ is $O(f)$
- Higher powers grow faster
  - n^r is $O(n^s)$ if $0 \leq r \leq s$

← Fastest growing term dominates a sum

- If f is $O(g)$, then $f + g$ is $O(g)$

  eg $an^4 + bn^3$ is $O(n^4)$

← Polynomial's growth rate is determined by leading term

- If f is a polynomial of degree d,

  then f is $O(n^d)$

Properties of the O notation

- f is $O(g)$ is transitive
  - If f is $O(g)$ and g is $O(h)$ then f is $O(h)$
- Product of upper bounds is upper bound for the product
  - If f is $O(g)$ and h is $O(r)$ then fh is $O(gr)$
- Exponential functions grow faster than powers
  - n^k is $O(b^n)$ $\forall$ $b > 1$ and $k \geq 0$
    eg n^{20} is $O(1.05^n)$
- Logarithms grow more slowly than powers
  - $\log_b n$ is $O(n^k)$ $\forall$ $b > 1$ and $k > 0$
    eg $\log_2 n$ is $O(n^{0.5})$

Important!

Properties of the O notation

- All logarithms grow at the same rate
  - $\log_b n$ is $O(\log_d n)$ $\forall$ $b, d > 1$

- Sum of first n r^{th} powers grows as the $(r+1)^{th}$ power
  - $\sum_{k=1}^{n} k^r$ is $\Theta(\ n^{r+1})$

$$eg \quad \sum_{k=1}^{n} i = \frac{n(n+1)}{2} \quad \text{is} \quad \Theta(\ n^2)$$

Polynomial and Intractable Algorithms

- **Polynomial Time complexity**
  - **An algorithm is said to be polynomial if it is $O(n^d)$ for some integer d**
  - **Polynomial algorithms are said to be efficient**
    - **They solve problems in reasonable times!**

- **Intractable algorithms**
  - **Algorithms for which there is no *known* polynomial time algorithm**
  - ***We will come back to this important class later in the course***

Analysing an Algorithm

- **Simple statement sequence**

  s_1; s_2; …. ; s_k

  - *O(1)* as long as *k* is constant

- **Simple loops**

```
for(i=0;i<n;i++) { s; }
```

  where `s` is *O(1)*

  - Time complexity is $n\,O(1)$ or $O(n)$

- **Nested loops**

```
for(i=0;i<n;i++)
    for(j=0;j<n;j++) { s; }
```

  - Complexity is $n\,O(n)$ or $O(n^2)$

This part is *O(n)*

Analysing an Algorithm

- **Loop index doesn't vary linearly**

```
h = 1;
while ( h <= n ) {
     s;
     h = 2 * h;
     }
```

- **h takes values $1, 2, 4, \ldots$ until it exceeds n**
- **There are $1 + \log_2 n$ iterations**
- **Complexity $O(\log n)$**

Analysing an Algorithm

- **Loop index depends on outer loop index**

```
for(j=0;j<n;j++)
    for(k=0;k<j;k++){
      s;
      }
```

  - **Inner loop executed**
    - **1, 2, 3, …., n times**

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$\therefore$ **Complexity** $O(n^2)$

Distinguish this case - where the iteration count increases (decreases) by a constant ← $O(n^k)$ from the previous one - where it changes by a factor ← $O(\log n)$